RAPid-Learn: A Framework for Learning to Recover for Handling Novelties in Open-World Environments.

Shivam Goel[1,*], Yash Shukla[1,*], Vasanth Sarathy[2], Matthias Scheutz[1] and Jivko Sinapov[1]

Abstract— We propose RAPid-Learn (Learning to Recover and Plan Again), a hybrid planning and learning method, to tackle the problem of adapting to sudden and unexpected changes in an agent's environment (i.e., novelties). RAPid-Learn is designed to formulate and solve modifications to a task's Markov Decision Process (MDPs) on-the-fly and is capable of exploiting domain knowledge to reason about any new dynamics caused by the environmental changes. This novelty information is reflected in its updated domain model. We demonstrate its efficacy by introducing a wide variety of novelties in a gridworld environment inspired by *Minecraft*, and compare our algorithm with transfer learning baselines from the literature. Our method is (1) effective even in the presence of multiple novelties, (2) more sample efficient than transfer learning RL baselines, and (3) robust to incomplete model information, as opposed to pure symbolic planning approaches.

I. INTRODUCTION

AI systems have shown exceptional performance in many "closed worlds" domains such as games [1] where the action space, state space, and the transition dynamics are fixed for duration of the task. Even minor changes in the environment dynamics, however, can lead to catastrophic results for closed-world agents [2]. To develop AI systems that can function efficiently in the real world and thus in "open-world" settings where sudden and unexpected changes (i.e., novelties) can occur [3], we need to relax closed-world assumptions and make agents robust to novel, unseen situations [4].

The need to adapt to unexpected environmental changes has prompted some to employ *Reinforcement Learning* (RL) techniques where learning is lifelong [5], [6] and non-stationary [7]. However, these approaches assume a continuous evolution of the environment and are less effective in open-world settings where changes can be abrupt and lead to catastrophic forgetting, thus requiring the agent to potentially re-learn the entire task. Symbolic planning approaches, on the other hand, involve an agent reasoning about sequences of predefined operators and optimizing the resultant action sequence that reaches a goal with high probability [8]. These approaches effectively plan over long horizons, and work well when the domain knowledge and planning operators are available and defined. However, these approaches are laborious to design and assume an accurate and complete model of

*(*equal contribution)*
[1] Department of Computer Science, Tufts University, Email: {shivam.goel, yash.shukla, matthias.scheutz, jivko.sinapov} @ tufts.edu
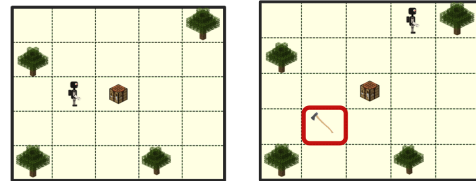[2] Smart Information Technology (SIFT), Email: vsarathy@sift.net

Fig. 1: Left: Pre-novelty domain showing an agent which can craft items by breaking *trees*. Right: Novelty induces an impasse which requires the agent to use the axe (shown in red) to break the *trees*.

the environment. They are often ineffective in evolving, non-stationary, and open-world environments, when the known model is incomplete.

In this paper we demonstrate that a symbiotic combination of RL-based and planning-based approaches can overcome the shortcomings of either approach and is thus desirable for changing environments(Figure 1 shows an example of a changing environment). However, integrating these paradigms is challenging, because the RL policies need to be abstracted in a way that is beneficial for symbolic planners. The challenges are best illustrated by recent approaches aimed at integrating learning and planning which have a limited set of state and action spaces, or are computationally expensive [9]. While employing RL to learn low-level policies for high-level plan operators can help to address the agent's incomplete knowledge in open worlds [10], [11], recent hybrid approaches are either not evaluated in open-world settings where plan failures are typical, or assume a fixed environmental configuration and a custom planner [12], making them applicable only to a subset of novelties.

Our proposed solution is RAPid-Learn, a method for recovering from fatal plan failures caused by novelties (sudden and unexpected changes that can result in an execution impasse) (refer Figure 1) by exploiting symbolic knowledge to perform sample-efficient explorations of novelties. To demonstrate its effectiveness, we compare the novelty adaptation success with state-of-the-art RL methods spanning hierarchical and transfer-learning approaches. A thorough evaluation of our agent in a Minecraft-inspired domain demonstrates the superiority of the proposed methods compared to RL agents.

II. RELATED WORK

The problem we address has been referred to as "open-world novelty accommodation" [3], [13]. Here, the aim is to detect and accommodate the novelty, without any prior

information of its dynamics.

Symbolic planning approaches have shown progress in adapting to unforeseen situations in the environment by exploiting existing domain knowledge [14], [15], [16]. However, they fail to explicitly accommodate the dynamics of the novelty, while focusing on open-world goals and instructions. Symbolic knowledge representations have also been used for plan recovery [17], with one key assumption of an accurate knowledge model. Updating the pre-conditions and effects of the agent's operators is effective in open-world settings [13], [18], [19], although the lack of model-free learning limits novelty handling to only those novelties that can be derived from the existing knowledge base. To bridge this gap, we propose accommodating the novelty by instantiating a model-free reinforcement learner.

Reinforcement learning approaches typically adapt to continuously evolving and non-stationary environments [20], [6], [21], [7]. Meta-RL attempts at adapting to multiple MDPs, and solving an unseen task drawn from the same distribution as the training tasks [22]. However, these approaches either consider gradually evolving environments or are suited for tasks drawn from the same distribution, and fail to adapt to sudden changes in the environment dynamics [23]. Unlike prior work, we propose a sample-efficient hybrid learner and planner, aimed at detecting and accommodating novelties, when plan execution fails.

Hybrid Planning and Learning approaches take advantage of a sub-symbolic reinforcement learning agent to aid a symbolic planner to address non-stationarity in open-world settings [12], [24]. RL sub-goal learner provides a robust sub-symbolic policy for each operator of a symbolic planner [10], [25], [26]. However, they assume an accurate knowledge representation and are not discussed in open-world settings. Other approaches that utilize an RL agent to accommodate the novelty [12] assume a stationary configuration of the initial state, require a custom planner, do not exploit existing domain knowledge for RL exploration, and do not work with function approximators, rendering a sample inefficient hybrid agent catering to only a subset of open-world novelties. To the best of our knowledge, our approach is the first that adapts to the sudden, unknown changes by instantiating a deep reinforcement learner and exploits domain knowledge to speed-up learning.

III. Preliminaries

A. Symbolic Planning

We assume that the agent starts with a domain knowledge, grounded using PDDL [27], defined as $\Sigma = \langle \mathcal{E}, \mathcal{F}, \mathcal{S}, \mathcal{O} \rangle$, where $\mathcal{E} = \{\varepsilon_1, \ldots \varepsilon_{|\mathcal{E}|}\}$ is a finite set of known entities within the environment. $\mathcal{F} = \{f_1(\odot), \ldots f_{|\mathcal{F}|}(\odot)\}, \odot \subset \mathcal{E}$ is a finite set of known predicates with their negations. Each predicate $f_i(\odot)$, along with its negation $\neg f_i(\odot)$, is contained in $\mathcal{F}$. $\mathcal{S} = \{s_1 \ldots s_{|\mathcal{S}|}\}$ is the set of symbolic states in the environment. $\mathcal{O}$ denotes the set of known action operators such that $\mathcal{O} = \{o_1, \ldots o_{|\mathcal{O}|}\}$. Each operator o_i is defined with a set of preconditions and effects, denoted $\psi_i, \omega_i \in \mathcal{F}$. The preconditions and effects ψ_i, ω_i of o_i indicate the predicates that must hold true (or false) before and after executing o_i, respectively. We define a planning task as $T = (\mathcal{E}, \mathcal{F}, \mathcal{O}, s_0, s_g)$, in which $s_0 \subset \mathcal{S}$, is the set of starting states and $s_g \subset \mathcal{S}$ is the set of goal states. The solution to the planning task T is an ordered list of operators, given by plan $\mathcal{P} = [o_1, \ldots o_{|\mathcal{P}|}]$.

B. Reinforcement Learning & Sub-Symbolic Executors

An episodic Markov Decision Process (MDP) $\tilde{M}$ is defined as a tuple $\langle \tilde{\mathcal{S}}, \tilde{\mathcal{A}}, \tilde{p}, \tilde{r}, \tilde{\gamma} \rangle$, where $\tilde{\mathcal{S}}$ is the set of sub-symbolic states, $\tilde{\mathcal{A}}$ is the set of actions, $\tilde{p}(\tilde{s}_{t+1} | \tilde{s}_t, \tilde{a}_t)$ is the transition function, $\tilde{r}(\tilde{s}_{t+1}, \tilde{a}_t, \tilde{s}_t)$ is the reward function and $\tilde{\gamma} \in [0, 1]$ is the discount factor. For each timestep t, the agent observes a state $\tilde{s}$ and performs an action $\tilde{a}$ given by its policy function $\pi_\theta(\tilde{a} | \tilde{s})$, with parameters θ. The agent's goal is to learn an *optimal policy* π^*, maximizing its discounted return $G_0 = \sum_{k=0}^{K} \tilde{\gamma}^k \tilde{r}(\tilde{s}'_k, \tilde{a}_k, \tilde{s}_k)$ until the end of the episode at timestep K.

At the sub-symbolic level, we define a set of action executors $\mathcal{X} = \{x_1, x_2, x_3, \ldots\}$. Inspired by the options framework in RL, an action executor x_i is represented by a tuple $\langle \mathcal{I}_{x_i}, \pi_{x_i}, \beta_{x_i} \rangle$, where $\mathcal{I}_{x_i} \subseteq \tilde{\mathcal{S}}$ is the initiation set, denoting the set of states when the action executor x_i is available for execution, and it follows a policy $\pi_{x_i} : \tilde{\mathcal{S}} \times \tilde{\mathcal{A}} \to [0, 1]$. $\beta_{x_i}(\tilde{s}_0, \tilde{s}) \in \{0, 1\}$ is the indicator variable with value 1 if x_i can be terminated at $\tilde{s}$ given it was initialised at $\tilde{s}_0$, and 0 otherwise.

C. Novelty

We define novelty as a completely new encounter for the agent, where the agent can neither derive the dynamics of the novelty using its cognitive abilities, nor through its previous experiences (given domain knowledge Σ) [13]. Formally, we define novelty as a tuple $\mathcal{N} = \langle \mathcal{E}', \mathcal{F}', \mathcal{S}', \mathcal{O}' \rangle$, where $\mathcal{E}'$ represents the set of novel entities in the environment such that $\mathcal{E}' \cap \mathcal{E} = \emptyset$. $\mathcal{S}'$ represents the set of novel states such that $\mathcal{S}' \cap \mathcal{S} = \emptyset$, and $\mathcal{O}'$ denotes the set of novel operators such that $\mathcal{O}' \cap \mathcal{O} = \emptyset$, $\mathcal{F}'$ is a set of novel predicates which are unknown to the agent. We assume that $\mathcal{N}$ transforms the domain knowledge from Σ to Σ' by including novelties which results in an execution impasse. The new domain Σ' can generate a planning solution, but the plan execution will result in a failure due to incomplete domain knowledge.

IV. Problem Formulation

We build upon our prior work [12] and formulate a framework for integrating planning and learning, where an integrated planning task enables us to ground operators in an MDP, specify goals symbolically, and realize action hierarchies. We define an *executor* for a given MDP $\tilde{M}_{x_i} = (\tilde{\mathcal{S}}_{x_i}, \tilde{A}_{x_i}, \tilde{p}_{x_i}, \tilde{r}_{x_i}, \tilde{\gamma}_{x_i})$ as a triplet $x_i = \langle \mathcal{I}_{x_i}, \pi_{x_i}, \beta_{x_i} \rangle$ where $\mathcal{I}_{x_i} \subseteq \tilde{\mathcal{S}}_{x_i}$ is an initiation set, $\pi_{x_i}(\tilde{a} | \tilde{s})$ is the probability of performing $\tilde{a}$, given current state $\tilde{s}$, and $\beta_{x_i}(\tilde{s}_0, \tilde{s}) \in \{0, 1\}$ is the indicator variable with value 1 if x_i can be terminated at $\tilde{s}$ given it was initialized at $\tilde{s}_0$, and 0 otherwise. An executor is simply an operator where the policy and termination condition depend on where it was initialized.
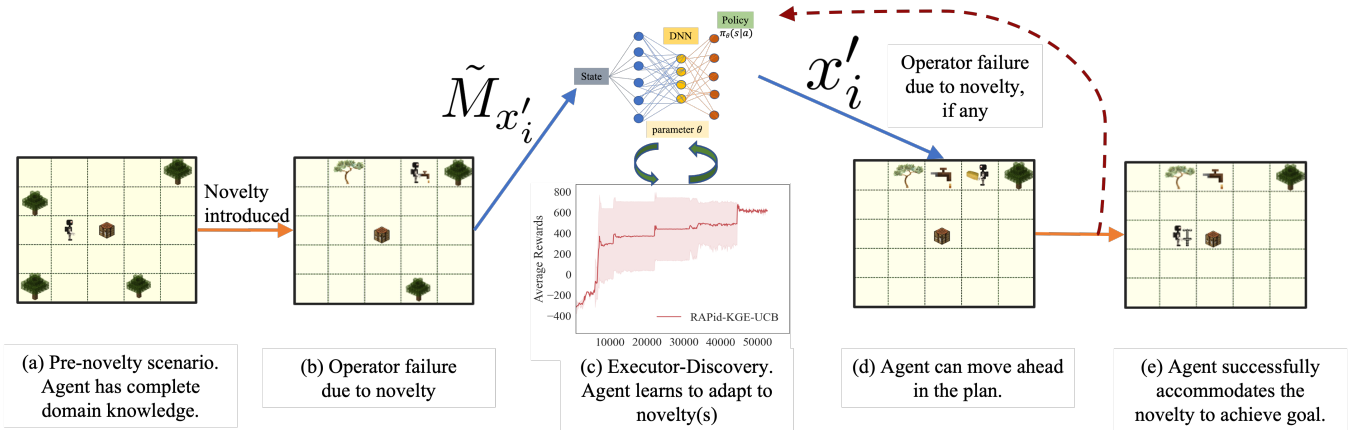
Fig. 2: Overview of the experimental setup.(a) Illustrates the NovelGridworlds domain in the pre-novelty scenario, showing an agent in a walled arena surrounded by *trees*, and facing a *crafting-table*. (b) Shows the scenario when the environment introduces the *rubber-tree-hard* novelty and the agent reaches an impasse.(c) *Executor-Discovery* phase when the agent accommodates the novelty using our approach RAPid-KGE-UCB. (d) Agent successfully proceeds in the plan. (e) Agent reaches the goal state of crafting a *pogo-stick*.

We define $\mathcal{X}$ as the set of all executors for the set of MDP $\mathcal{M}$.(Section III-B).

Definition 1. *[12] (Integrated Planning Task) Formally, an Integrated Planning Task (IPT) is $\mathcal{T} = \langle T, \mathcal{M}, d, e \rangle$ where $T = \langle \mathcal{E}, \mathcal{F}, \mathcal{O}, s_o, s_g \rangle$ is a STRIPS task, $\mathcal{M}$ is the set of MDPs. A detector function $d : \tilde{\mathcal{S}} \mapsto \mathcal{S}$ determines a symbolic state for a given sub-symbolic MDP state, and an executor function $e : \mathcal{O} \mapsto \mathcal{X}$ maps an operator to an executor.*

As described in Sec. III-A, let $\mathcal{P}$ be the solution of a planning task $T(\mathcal{E}, \mathcal{F}, \mathcal{O}, s_0, s_g)$. We assume that for each operator $o \in \mathcal{O}$, its executor $e(o)$ accurately maps to o; that is, for every $o \in O$, $\mathcal{I}_{e(o)} \supseteq \{\tilde{s}_0 \in \tilde{\mathcal{S}}_{e(o)} : d(\tilde{s}_0) \supseteq \psi_o\}$ and

$$\beta_{e(o)}(\tilde{s}_0, \tilde{s}) = \begin{cases} 1 & \text{if } (d(\tilde{s}) \supseteq \omega_o \vee ((d(\tilde{s}) \supseteq \omega_{\hat{o}}) \forall \hat{o} \in \hat{\mathcal{O}}) \\ & \wedge \exists \hat{\mathcal{P}} \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

The executor $e(o)$ reaches a termination state $\tilde{s}$ when it satisfies the effects of the operator o $(d(\tilde{s}) \supseteq \omega_o)$, or if it satisfies the effects of all subsequent operators $\hat{o} \in \hat{\mathcal{O}}$ in the plan $\mathcal{P}$ $((d(\tilde{s}) \supseteq \omega_{\hat{o}}) \forall \hat{o} \in \hat{\mathcal{O}})$, where the preconditions of $\hat{o}$ contain the effects of o $(\psi_{\hat{o}} \supseteq \omega_o)$. Also, ensuring a planning solution $\hat{\mathcal{P}}$ to the task $\hat{\tilde{T}} = (\mathcal{E}, \mathcal{F}, \mathcal{O}, d(\tilde{s}), s_g)$ exists. A *solution* to the IPT $\mathcal{T}$ is an ordered list of executors $[x_1, \ldots x_{|\mathcal{X}|}] \in \mathcal{X}$ having the above mentioned properties. A *planning solution* to an IPT $\mathcal{T}$ is the ordered list of operators given by the $\mathcal{P} = [o_1, \ldots o_{|\mathcal{P}|}]$. Executing the ordered list of executors in the sub-symbolic space will yield a final state $\tilde{s}$ such that $d(\tilde{s}) \supseteq s_g$, thus reaching the goal state s_g. $\mathcal{T}$ is solvable if a solution exists and plannable if a planning solution exists.

A. The Executor Discovery Problem

We define a stretch-Integrated Planning Task (Stretch-IPT), that captures difficult but achievable goals – those for which missing executors must be discovered.

Definition 2. *(Stretch-IPT). A Stretch-IPT $\tilde{\mathcal{T}}$ is an IPT $\mathcal{T}$ for which a solution exists, but a planning solution does not.*

A novelty introduced by the environment results in incomplete domain knowledge (Section III-C), causing an execution impasse due to operator failure. We are interested in finding a solution to the stretch-IPT, specifically study how to automatically generate executors on-the-fly to solve the execution impasse.

Definition 3. *(Executor Discovery Problem). Given a stretch-IPT $\tilde{\mathcal{T}} = \langle T', \mathcal{M}', d', e' \rangle$ with $T' = \langle \mathcal{E} \cup \mathcal{E}', \mathcal{F} \cup \mathcal{F}', \mathcal{O} \cup \mathcal{O}', s'_o, s_g \rangle$, construct a set of executors $\{x'_1, \ldots, x'_m\} \in \mathcal{X}$ for the set of failed operators $\{o_1, \ldots, o_m\} \in \mathcal{O}$ such that the stretch-IPT $\tilde{\mathcal{T}}$ is solvable, with the executor function:*

$$e'(o_i) = x'_i, o_i \notin \mathcal{O} \quad (2)$$

i.e., we find an executor whose operator does not exist in $\mathcal{O}$.

V. RAPID-LEARN

Below we describe a running example followed by a detailed description of our approach.

A. Description of the domain

Figure 2a shows a gridworld representation of a *Minecraft* inspired domain. An agent is shown enclosed in a walled arena that contains *trees* and a *crafting-table* where more complex items can be crafted. The agent can obtain *tree-log* by breaking *trees*. *Tree-logs* can be crafted into *planks*, which in turn can be crafted into *sticks*. A combination of *planks* and *sticks* can be crafted into a *tree-tap* when facing the *crafting-table*. When the *tree-tap* is *selected* next to a *tree*, *rubber* can be extracted. Finally, a combination of *sticks*, *planks* and *rubber* crafts a *pogostick*.

Algorithm 1 *RAPid-Learn($\mathcal{T}$)*

1: Input: $\mathcal{T} = (T, \mathcal{M}, d, e)$: Integrated Planning task
2: $T = \langle \mathcal{E}, \mathcal{F}, \mathcal{S}, \mathcal{O}, s_0, s_g \rangle; \mathcal{X} \leftarrow \emptyset$
3: $\mathcal{P}$ = **Planner**$(\mathcal{T})$ $\quad \{\mathcal{P} = \{o_1, o_2, ..., o_{|\mathcal{P}|}\}\}$
4: **for** $o_i \in \mathcal{P}$ **do**
5: $\quad \mathcal{X} \leftarrow \mathcal{X} \cup e(o_i)$
6: **end for**
7: **for** $o_i \in \mathcal{P}$ **do**
8: $\quad$ Success = **Execute** (o_i)
9: $\quad$ **if** $\neg$Success **then**
10: $\quad\quad$ **if** $x_i' \in \mathcal{X}$ **then**
11: $\quad\quad\quad$ **Execute**(x_i')
12: $\quad\quad$ **else**
13: $\quad\quad\quad x_i' \leftarrow$ ***Discover-Executor****$(o_i, \mathcal{P}, \tilde{\mathcal{T}}, k)$; $\mathcal{X} \leftarrow \mathcal{X} \cup \{x_i'\}$

14: $\quad\quad\quad$ **Execute**(x_i')
15: $\quad\quad$ **end if**
16: $\quad$ **end if**
17: **end for**

B. Running Example

Let us consider a novelty scenario in which a *rubber-tree* ($\varepsilon_1 \in \mathcal{E}'$) appears in the environment (Figure 2b). The novel environment also provides a new action *place-tree-tap*, and a novel hierarchical action *approach-rubber-tree*. The agent can extract *rubber* by placing a *tree-tap* only in front of the *rubber-tree*. This results in the failure of operator o_i =(extract-rubber) . To overcome this execution impasse, an MDP $\tilde{M}_{x_i}$ (Figure 2(c)) is instantiated, whose solution is the learned executor x_i' which is mapped to the failed operator o_i for future use. Action executor x_i''s policy $\pi_{x_i'}$ will consists of steps that involve the agent to stand one block away from the *rubber-tree*, place the *tree-tap* next to the *rubber-tree* and then use the failed operator *extract-rubber*. After the agent successfully executes x_i'(Figure 2 (d)), it switches back to original plan to craft a *pogostick*(Figure 2(e)).

C. Running in an impasse

As shown in algorithm 1, the agent starts with an integrated planning task $\mathcal{T}$ as an input. Using the domain knowledge grounded through PDDL [27], the agent uses MetricFF [28] planner to generate a planning solution (plan $\mathcal{P}$) to the IPT $\mathcal{T}$ (line 3). The agent then executes this plan (line 4-8) in a novel environment, defined by a stretch-IPT $\tilde{\mathcal{T}}$. We assume that one of the operators will fail in the novel environment, resulting in an execution impasse. The impasse occurs if the known effects of the operator are not true after the execution of the operator in the environment. Once the agent detects an execution impasse, it checks if it has a corresponding sub-symbolic action executor x_i' for the failed operator o_i (line 10-12). If the executor does not exist, the agent enters *executor-discovery* mode (line 13). We now describe how the agent instantiates and solves an executor discovery problem to succeed in the execution impasse induced by the novelty.

Algorithm 2 *Discover-Executor*$(o_i, \mathcal{P}, \tilde{\mathcal{T}}, k) \rightarrow x_i'$

1: $\tilde{\mathcal{T}} = \langle T', \mathcal{M}', d', e' \rangle; T' = \langle \mathcal{E} \cup \mathcal{E}', \mathcal{F} \cup \mathcal{F}', \mathcal{O} \cup \mathcal{O}', s_o', s_g \rangle$
2: Hyperparameters : $\rho_0, \epsilon_0, c, \mu, e_{max}$
3: $\tilde{\mathcal{S}}_{x_i'}'$: set of sub-symbolic states; $\tilde{\mathcal{A}}_{x_i'}'$: set of actions;
4: $\beta_{x_i'}'$: indicator function for executor x_i' as defined in 1
5: $\tilde{\mathcal{S}}_0 \in \tilde{\mathcal{S}}_{x_i'}'$ set of initial states; $\Delta \leftarrow \emptyset$: set of actions to bias
6: $\mathcal{E}_{novel} \leftarrow \{\mathcal{E}'\} \setminus \{\mathcal{E}\}$: set of novel entities
7: $\tilde{\mathcal{A}}_{novel} \leftarrow \{\tilde{\mathcal{A}}_{x_i'}'\} \setminus \{\tilde{\mathcal{A}}\}$: set of novel actions
8: $\Delta \leftarrow \Delta \cup \{\tilde{\mathcal{A}}_{novel}, o_i\}$
9: StateSet $\leftarrow \emptyset$, ActSet $\leftarrow \emptyset$, DoneSet $\leftarrow \emptyset$, RewSet $\leftarrow \emptyset$
10: $\mathcal{S}_r \leftarrow$ ***PlannableStateGenerator****$(\Sigma, \mathcal{P}, o_i)$
11: $e \leftarrow 1$: episode number
12: **while** $e < e_{max}$ **do**
13: $\quad \tilde{s}_0' \leftarrow:$ ***ReachFailedOperator****$(\tilde{\mathcal{T}})$: initial state
14: $\quad done \leftarrow false, t \leftarrow 1$
15: $\quad$ **if** $k \neq EG$ & $uniformRandomNo(0,1) < \rho_0$ **then**
16: $\quad\quad \varepsilon \leftarrow random(\mathcal{E}_{novel})$; $p \leftarrow Planner(\varepsilon, \tilde{s}_0')$
17: $\quad\quad \tilde{s}_t' \leftarrow execute(\tilde{s}_0', p)$
18: $\quad$ **end if**
19: $\quad A_p \leftarrow \{\pi(\tilde{a}_i'|\tilde{s}_t') \forall \tilde{a}_i' \in \tilde{\mathcal{A}}_{x_i'}'\}$
20: $\quad$ **while** $\neg done \ \vee \ t < U$ **do**
21: $\quad\quad$ **if** $uniformRandomNo(0,1) < \epsilon_0$ **then**
22: $\quad\quad\quad$ **if** $k = KGE\text{-}UCB$ **then**
23: $\quad\quad\quad\quad$ **for** $\tilde{a}' \in \tilde{\mathcal{A}}_{x_i'}'$ **do**
24: $\quad\quad\quad\quad\quad A_p[\tilde{a}'] \leftarrow A_p[\tilde{a}'] \begin{cases} +c\sqrt{\frac{\ln t}{N_t(\tilde{a}')}} & if \tilde{a}' \in \Delta \\ -c\sqrt{\frac{\ln t}{N_t(\tilde{a}')}} & if \tilde{a}' \notin \Delta \end{cases}$
25: $\quad\quad\quad\quad$ **end for**
26: $\quad\quad\quad$ **else if** $k = KGE\text{-}UAB$ **then**
27: $\quad\quad\quad\quad$ **for** $\tilde{a}' \in \tilde{\mathcal{A}}_{x_i'}'$ **do**
28: $\quad\quad\quad\quad\quad A_p[\tilde{a}'] \leftarrow \begin{cases} \frac{(\mu - 1 + \Sigma\Delta)\tilde{a}'}{\mu\Sigma\Delta} & if \ \tilde{a}' \in \Delta \\ \tilde{a}'/\mu & if \ \tilde{a}' \notin \Delta \end{cases}$
29: $\quad\quad\quad\quad$ **end for**
30: $\quad\quad\quad$ **end if**
31: $\quad\quad$ **end if**
32: $\quad\quad \tilde{a}_t' \leftarrow$ arg-max(A_p)
33: $\quad\quad \tilde{s}_{t+1}' \leftarrow execute(\tilde{a}_t')$
34: $\quad\quad done, r \leftarrow r(\mathcal{S}_r, \tilde{s}_t', \tilde{a}_t, \tilde{s}_{t+1}')$
35: $\quad\quad$ DoneSet $\leftarrow$ DoneSet$\cup\{done\}$, StateSet $\leftarrow$ StateSet$\cup\{\tilde{s}_t'\}$

36: $\quad\quad$ RewSet $\leftarrow$ RewSet $\cup \{r\}$, ActSet $\leftarrow$ ActSet $\cup A_p$
37: $\quad\quad t \leftarrow t + 1, \tilde{s}_{t+1}' \leftarrow \tilde{s}'$
38: $\quad$ **end while**
39: $\quad e \leftarrow e + 1$
40: $\quad \pi \leftarrow$ *Update-Network*(StateSet, ActSet, RewSet)
41: $\quad$ **if** *Converge*(DoneSet, RewSet) $> \eta$ **then**
42: $\quad\quad \pi_{x_i'}^c \leftarrow \pi, x_i' \leftarrow \langle \tilde{S}_0', \pi_{x_i'}^c, \beta_{x_i'} \rangle$
43: $\quad\quad$ **return** x_i'
44: $\quad$ **end if**
45: **end while**
46: $x_i' \leftarrow \langle \tilde{S}_0', \pi_{x_i'}, \beta_{x_i'} \rangle$
47: **return** x_i'

D. Executor-Discovery

To find a solution to the stretch-IPT $\tilde{\mathcal{T}}$, the agent needs to discover an executor x_i' that will succeed through the impasse caused by the novelty. The executor discovery process is described in Algorithm 2. The agent instantiates an online reinforcement learner over the episodic MDP $\tilde{M}_{x_i'} = (\tilde{\mathcal{S}}_{x_i'}', \tilde{A}_{x_i'}', \tilde{p}_{x_i'}', \tilde{r}_{x_i'}, \tilde{\gamma}_{x_i'})$ the first time it encounters the failed

operator. This MDP consists of the set of sub-symbolic states $\tilde{\mathcal{S}}'_{x'_i}$ and actions $\tilde{\mathcal{A}}'_{x'_i}$ (sub-symbolic pre-novelty actions, sub-symbolic novel actions, and all the symbolic operators mapped to action executors). A sparse reward function is generated on-the-fly to guide the agent to discover the set of plannable states $\mathcal{S}_r$ from where it can reach the goal state s_g.

1) Discovery of Plannable States: With the pre-novelty domain knowledge, the agent accumulates the set of preconditions Ψ and effects Ω of all known operators. The agent then generates a set of plannable states $\mathcal{S}_r$, which contains: 1) the states that satisfy the effects ω_{o_i} of the failed operator o_i and 2) the states that satisfy the effects of all subsequent operators $\hat{o} \in \hat{\mathcal{O}}$ in the plan $\mathcal{P}$ $((d(\tilde{s}) \supseteq \omega_{\hat{o}}) \forall \hat{o} \in \hat{\mathcal{O}})$, where the preconditions of $\hat{o}$ contain the effects of o $(\psi_{\hat{o}} \supseteq \omega_o)$.[1] In each episode, the agent computes a plan from an initial configuration of the environment, and carries out this plan until it reaches the failed operator (Algorithm 2 - line 12). The agent is rewarded with a positive reward ϕ_1, if it reaches a state in $\mathcal{S}_r$ within a predetermined number of timesteps U, and a plan from this state to the goal state s_g exists. However, in some cases the agent can reach a state in $\mathcal{S}_r$ but the planning solution from this state to the goal state s_g does not exist (because of irreversible actions taken by the agent). To prevent this failure, a negative reward ϕ_2 is given, and the episode terminates. The agent gets a unit negative reward for all other timesteps. Formally, the reward function is defined as:

$$\mathrm{r}\left(\mathcal{S}_r, \tilde{s}'_t, \tilde{a}'_t, \tilde{s}'_{t+1}\right) = \begin{cases} \phi_1 & if\ \ d(\tilde{s}'_{t+1}) \subset \mathcal{S}_r \wedge \exists \hat{\mathcal{P}} \\ -\phi_2 & if\ \ d(\tilde{s}'_{t+1}) \subset \mathcal{S}_r \wedge \nexists \hat{\mathcal{P}} \\ -1 & \text{otherwise} \end{cases}$$

2) Knowledge-guided-exploration: To guide the agent to explore efficiently and utilize domain knowledge, we employ knowledge-guided exploration. We describe two approaches:

Knowledge-guided curriculum learning: Learning to solve complex problems may require extensive interactions with the environment. Knowledge obtained through simpler sub-tasks can be utilized to reduce the exploration of complex tasks [29]. *knowledge-guided curriculum learning* enables the agent to solve simpler sub-tasks, thereby increasing the probability of the agent landing into a plannable state. With a probability ρ, the agent is provided a curriculum to reach a novel state (line 14-16, Alg. 2). Randomly selecting a novel entity from the set of novel entities, the agent utilizes the planner to reach this novel entity (line 15). This procedure enables the agent to begin exploring from a promising initial state [30]. The agent is given the state transitions $(\tilde{s}'_t, \tilde{a}'_t, \tilde{s}'_{t+1})$ from its initial state to this promising state.

Knowledge-guided action biasing: In *knowledge-guided action biasing*, during exploration the agent biases a subset of actions through domain knowledge. The subset of the actions to bias Δ consists of; (1) novel actions, and (2) failed action operators (lines 6-7, Algorithm 2). With a probability ϵ_0, the probability of selecting these actions is

[1]Details are not central to the paper, but our specific algorithm is in GitHub Report Section 2

bumped up when the agent explores the environment (lines 20-26). We develop two methods of action probability bumping, Upper Confidence Bounds (KGE-UCB), and Uniform Action Biasing (KGE-UAB). In KGE-UCB, we bump up the probability of actions in the set Δ (A_p, line 18) inversely proportional to square root of the number of times the action was executed prior to the current timestep [31] (lines 21-24). This helps in selecting those actions that were tried the least, enabling the agent to visit new states and increasing the possibility of reaching the plannable states in $\mathcal{S}_r$. In KGE-UAB, we uniformly bump up the selection probability of all the actions in the set Δ (lines 25-29). Both these methods are an extension of the ϵ-greedy exploration (EG) strategy. Contrary to EG, they exploit domain knowledge to perform efficient exploration.

The rationale behind using knowledge-guided-exploration in such a way follows two assumptions: 1) The agent should use domain knowledge to guide exploration; and 2) The novelties are the reason why the agent gets stuck in an impasse.

The agent continues updating its policy until it reaches the maximum permitted episodes (e_{max}) (lines 31-39) or if it converges to a policy $\pi^c_{x'_i}$ defined by a pre-determined success rate convergence threshold η (lines 40-42). The agent then exits the executor discovery mode and continues with its original plan $\mathcal{P}$.

E. Recovery

Once the agent learns a policy $\pi^c_{x'_i}$ to succeed at the impasse, it is added as an action executor x'_i (Alg 2 lines 41,45) to the set of action executors $\mathcal{X}$ (line 10, Alg 1). The agent executes x'_i, whenever the operator o_i fails. This framework is effective even in multiple novelty settings, where more than one operators fail.

VI. EVALUATION AND RESULTS

A. NovelGridworlds Domain

We evaluate RAPid-Learn on a $[12 \times 12]$ NovelGridworlds domain [32], a gridworld crafting problem inspired by Minecraft (agent's task in the domain is described in Section V-A).

The domain was chosen for two key reasons: 1) it provides a complicated task that involves a sequential set of actions to reach the goal state. If the agent misuses its resources, it will not succeed in the task; and 2) the domain is designed specifically for open-world problem solving, enabling us to create and experiment with a variety of novelties [32].

B. Experimental Setup

Our setup (Figure 2) is designed to evaluate: (1) **Novelty accommodation**: The agent's ability to solve the impasse, regardless of the nature of novelty. For this, we evaluate RAPid-Learn on a variety of novel scenarios (Table I). The scenarios comprise of adding a novel entity, and/or, a novel operator in the agent's environment. In some cases, upon adding the novelty, the agent has to just learn to use the novel operator and the entity, while in others, it

Novelty Name	Entity ($\mathcal{E}'$)	Operator ($\mathcal{O}'$)	Dynamics ($\mathcal{S}'$)
axe-to-break (ATB) [easy/hard]	*axe*	*select-axe, approach-axe[hard]*	break *tree* only if holding *axe*; *[easy]* *axe* present in *inventory*; *[hard]* *axe* present in environment
fire-crafting-table (FCT) [easy/hard]	*fire, water*	*select-water; spray, approach-water[hard]*	*crafting-table* set on fire; need to *spray water* to access *crafting-table*; *[easy]* *water* present in *inventory* *[hard]* *water* present in the environment
rubber-tree (RT) [easy/hard]	*rubber-tree*	*place-tree-tap, approach-rubber-tree[hard]*	*[easy]* *rubber* can only be extracted when facing *rubber-tree*; *[hard]* need to place *tree-tap* in front of a *rubber-tree* to extract *rubber*.
scrape-plank	-	*scrape-plank*	cannot obtain *tree-log* by breaking *tree*, can only scrape planks from *tree*

TABLE I: Novelties Descriptions: A novelty changes the environment by adding new entities, operators, and dynamics.

Agent	Time to adapt (timesteps)	Post-novelty performance (success rate)
	Mean±SD	Mean±SD
Rubber Tree (RT) - Hard		
RAPid-KGE-UCB	$(2.85 \pm 2.15) \times 10^4$	0.95 ± 0.01
RAPid-KGE-UAB	$(3.28 \pm 1.51) \times 10^4$	0.93 ± 0.03
RAPid-EG	$(3.73 \pm 2.08) \times 10^4$	0.91 ± 0.04
Actor-Critic Transfer	did not converge	N.A
Policy-Reuse	did not converge	N.A
Axe to Break (ATB) + Fire Crafting Table (FCT) - Easy		
RAPid-KGE-UCB	$(4.02 \pm 1.82) \times 10^3$	0.97 ± 0.01
RAPid-KGE-UAB	$(4.56 \pm 3.43) \times 10^3$	0.97 ± 0.02
RAPid-EG	$(6.93 \pm 4.41) \times 10^3$	0.98 ± 0.01
Actor-Critic Transfer	$(17.6 \pm 5.13) \times 10^4$	0.93 ± 0.02
Policy-Reuse	$(8.08 \pm 2.38) \times 10^5$	0.92 ± 0.04
Axe To Break (ATB) - Hard		
RAPid-KGE-UCB	$(2.35 \pm 1.27) \times 10^3$	0.95 ± 0.01
RAPid-KGE-UAB	$(13.0 \pm 4.89) \times 10^2$	0.85 ± 0.07
RAPid-EG	$(14.9 \pm 8.65) \times 10^2$	0.84 ± 0.03
Actor-Critic Transfer	$(9.80 \pm 4.75) \times 10^4$	0.90 ± 0.05
Policy-Reuse	$(5.57 \pm 2.59) \times 10^5$	0.92 ± 0.04
Fire Crafting Table (FCT) - Hard		
RAPid-KGE-UCB	$(8.61 \pm 4.73) \times 10^3$	0.98 ± 0.00
RAPid-KGE-UAB	$(4.40 \pm 1.96) \times 10^3$	0.99 ± 0.00
RAPid-EG	$(5.12 \pm 2.84) \times 10^3$	0.98 ± 0.00
Actor-Critic Transfer	$(16.7 \pm 3.99) \times 10^4$	0.90 ± 0.04
Policy-Reuse	$(7.90 \pm 2.70) \times 10^5$	0.94 ± 0.03
Scrape Plank (SP)		
RAPid-KGE-UCB	$(6.82 \pm 3.83) \times 10^3$	0.86 ± 0.03
RAPid-KGE-UAB	$(2.01 \pm 1.01) \times 10^3$	0.87 ± 0.03
RAPid-EG	$(17.2 \pm 7.38) \times 10^2$	0.96 ± 0.01
Actor-Critic Transfer	$(6.86 \pm 3.89) \times 10^4$	0.93 ± 0.03
Policy-Reuse	$(19.3 \pm 9.76) \times 10^4$	0.94 ± 0.03

TABLE II: Results of each agent's performance on 5 novel scenarios. In all the novelties, RAPid-Learn consistently outperforms baseline approaches. (Best performing agent highlighted.)

has to invent a novel predicate to solve the impasse. This comprehensive list, though non-exhaustive, captures many aspects of the robustness of the agent[2](2) **Sample Efficiency**: The number of interactions taken by the agent to solve the impasse. To compare sample efficiency, we evaluate RAPid-Learn's three variations, namely, 1) RAPid-KGE-UCB: RAPid-Learn with knowledge-guided-exploration using upper confidence bounds; 2) RAPid-KGE-UAB: RAPid-Learn with knowledge-guided-exploration using uniform action biasing; 3) RAPid-EG: RAPid-Learn with ϵ-greedy exploration function, and compare these with two baselines.

1) Baselines: Existing methods in integrating planning and learning are aimed at either generating policies for the high-level operators [10] or at formulating operators to tackle the change in the environment [12]. Approaches that generate policies for high-level operators are not discussed in open-world novelty settings, and others that formulate operators [12] are not robust to varying environment configurations and numeric predicates. We therefore compare the performance of our architecture against policy reuse [33] and actor-critic transfer learning [34]. For each transfer learning baseline[3], we pre-train the agent (until convergence:$> 96\%$ success rate for last 100 episodes) using dense reward shaping, where choosing the right action at the right time is given intermediate reward. During pre-novelty training, the observation and the action spaces are extended with place-holder elements to accommodate the introduced novelties that can extend the shape of these spaces. On the pre-trained expanded model, we perform transfer learning using two approaches: Policy reuse [33] transfers the learned policy and Actor-critic transfer [34] transfers the policy and value function through PPO [35].

C. Results

We evaluate 3 versions of RAPid-Learn along with 2 baselines on 5 novel scenarios (shown in Table II). Time

to adapt is the number of time steps each agent takes to adapt to a novelty (learned policy is frozen according to the pre-defined convergence criteria[4]). Post-novelty performance is the success rate by evaluating each agent on 100 episodes. In each episode we set a budget to the number of time steps the agent is given to execute the learned executor (if the budget is used up, we report an unsuccessful episode and assign 0 score to that episode). A score of 1 is recorded for each successful episode. After running 10 independent trials with different random seeds, we report the mean and standard deviations.

Table II shows that our framework performs better at adapting to all the novel scenarios and time to adapt is significantly lower than the baselines. In some of the cases, the baselines do not even converge to find a solution. Fig 3 compares the learning curves of different types of *knowledge-guided-exploration* approaches described in this paper.

1) Statistical Significance: To demonstrate that the average success rate of RAPid-Learn is consistently higher than the baseline approaches, we perform an unpaired t-test [36]. The data is generated from 10 trials for each method, and in each trial, the performance of 100 post-novelty episodes is measured. For the experiment, we consider a confidence

(a) Rubber Tree Hard   (b) ATB+FCT Easy   (c) Axe to Break Hard   (d) Fire Crafting Table Hard

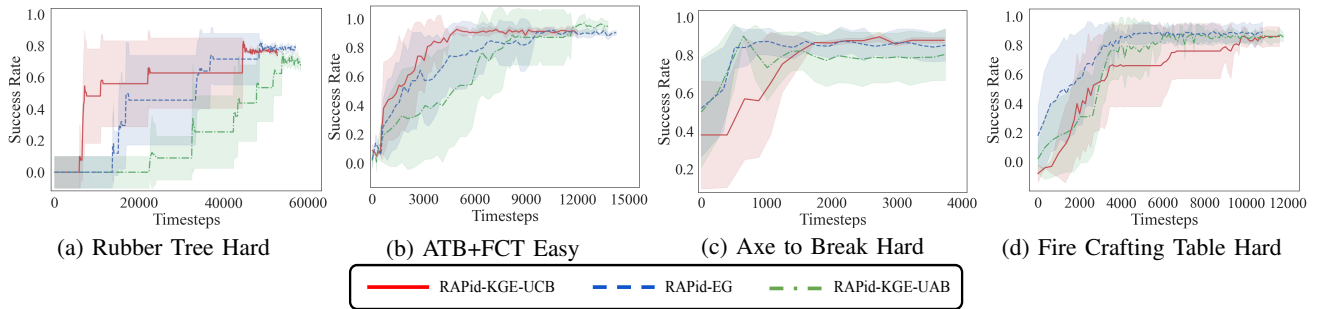RAPid-KGE-UCB — — — RAPid-EG — · — RAPid-KGE-UAB

Fig. 3: These plots illustrate the performance of the proposed RAPid-Learn methods during the *Executor-Discovery* phase of the associated novelty. The baseline methods take orders of magnitude more timesteps to converge, and are shown in GitHub Report Section 3.

interval of 95% and evaluate the t-value between the best performing RAPid-Learn approach and the two transfer learning baseline approaches (Actor-critic transfer , Policy reuse). Table III shows the results of the unpaired t-test. Thus, through the results, we see that our proposed approach, RAPid-Learn has a consistent performance in the post-novelty success rate. The results are always statistically significant, except in the case of the *Scrape Plank* novelty. In all the novel scenarios, RAPid-Learn has a much more sample efficient performance. Thus, RAPid-Learn not only achieves a better success rate, but also adapts to novel scenarios efficiently.

Methods	p-value	Statistical Significance
Axe to Break (ATB) + Fire Crafting Table (FCT) - Easy		
RAPid-KGE-UCB ↔ Actor-critic	0.0062	Yes
RAPid-KGE-UCB ↔ Policy Reuse	0.0012	Yes
Axe To Break (ATB) - Hard		
RAPid-KGE-UCB ↔ Actor-critic	0.00336	Yes
RAPid-KGE-UCB ↔ Policy Reuse	0.0012	Yes
Fire Crafting Table (FCT) - Hard		
RAPid-KGE-UAB ↔ Actor-critic	< 0.0001	Yes
RAPid-KGE-UAB ↔ Policy Reuse	< 0.0001	Yes
Scrape Plank (SP)		
RAPid-KGE-EG ↔ Actor-critic	0.0077	Yes
RAPid-KGE-UCB ↔ Policy Reuse	0.0608	Yes

TABLE III: Results of unpaired t-test between best performing RAPid-Learn and transfer learning baselines

D. Discussion

1) Dedicated Operator Failure: In these set of experiments, we aim to find out: *Can the agent learn to succeed when an operator failure occurs due to one novelty introduced by the environment?* The novel scenarios corresponding to this case are *axe-to-break hard*, *fire-crafting-table-hard*, and *scrape-plank* [5] (see Table I). The results in Table II and the learning curves in 3(c), 3(d) show that RAPid-KGE-UCB and RAPid-KGE-UAB converge to a post-novelty performance of 96%, beating the baselines by 2 orders of magnitude. The RAPid-Learn agent achieves a high success rate in the post-novelty performance in all the novel scenarios.

[5]GitHub Report Section 1 details out description of each novelty.

2) Explicit Multiple Operator Failures: In these set of experiments, we aim to find out: *Can the agent learn to succeed when multiple independent operator failures occur due to multiple novelties introduced by the environment?* The environment introduces two novelties simultaneously, namely, *fire-crafting-table easy* and *axe-to-break easy* (*ATB+FCT-Easy*). The effects of these novelties are independent in nature, i.e. the agent's performance over the *fire-crafting-table easy* novelty does not depend on its performance of solving the *axe-to-break easy* novelty. Thus, the agent learns two independent executors, one for each novelty. The results in Table II that RAPid-learn performs better than the baselines, achieving a post-novelty success performance of 97%. Fig 3(b) compares different RAPid-Learn approaches.

3) Implicit Multiple Operator Failures: In these set of experiments, we aim to find out: *Can the agent learn to succeed when its learned action executor further breaks an operator in the environment?* When the agent learns to adapt the *rubber-tree hard* novelty by extracting *rubber* in its inventory, it fails in executing the next operator in its original plan (as it can no longer break the novel unbreakable *rubber-tree*), running into another impasse. The agent can complete the task only after it adapts to the multiple implicit novelties in the environment. Table II shows that the baselines don't even converge, whereas RAPid-Learn learns to adapt in about 28000 time steps. Fig 3(a) compares different RAPid-Learn approaches.

In all the novel scenarios in our experiments, the agent cannot succeed at the impasse by exploring the known predicates and can only succeed when it explores the environment at a sub-symbolic level. For example, in the *rubber-tree-hard* novelty the agent's domain knowledge does not have any predicate for standing one block away from the *tree* entity for performing the *place* operator. Sub-symbolic learning materializes this notion through its sequence of actions in its policy, and in turn, in its action executor.

VII. Conclusion & Future Work

We proposed a novel hybrid planning-deep RL learning-based approach for open-world tasks. It handles unanticipated changes to the task environment on-the-fly which either method cannot sufficiently handle. Our proposed method

utilizes relevant domain knowledge to perform knowledge-guided-exploration in the novel environment and efficiently learns a robust novelty-handling policy mapped onto an action executor. A rigorous evaluation of our domain-independent method in five novel scenarios demonstrated the significant performance improvement compared to state-of-the-art transfer learning approaches. Our approach successfully accommodates novelties even in scenarios where baselines fail to converge.

While our current implementation only handles cases where novelties lead to plan execution failure or, more specifically, execution impasses, we are working on extensions that can also handle other types of open-world novelties (beneficial, detrimental, irrelevant). Moreover, our approach assumes that a plan always exists, and we are working on extensions in which the agent need to find a solution when the plan failure occurs. Furthermore, while we have demonstrated the approach in a fully observable, deterministic environment, nothing hinges on these assumptions and thus this approach can be generalized to incorporate stochasticity and partial observability. A limitation of our approach is we assume a fixed goal state. It will be interesting to extend this approach where we relax these assumptions. Finally, we are working on extensions to abstract the learned executor in the form of a symbolic operator, which is non-trivial when using function approximators.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] X. Qu, Z. Sun, Y. S. Ong, A. Gupta, and P. Wei, "Minimalistic attacks: How little it takes to fool deep reinforcement learning policies," *IEEE Transactions on Cognitive and Developmental Systems*, 2020.

[3] P. Langley, "Open-world learning for radically autonomous agents," in *AAAI*, vol. 34, no. 09, 2020, pp. 13 539–13 543.

[4] M. Kejrivwal and S. Thomas, "Reinforcing robustness of ai agents to open-world novelty: Challenges and roadmap," *Academia Letters*, p. 2, 2021.

[5] D. Abel, Y. Jinnai, S. Y. Guo, G. Konidaris, and M. Littman, "Policy and value transfer in lifelong reinforcement learning," in *ICML*. PMLR, 2018, pp. 20–29.

[6] E. Lecarpentier, D. Abel, K. Asadi, Y. Jinnai, E. Rachelson, and M. L. Littman, "Lipschitz lifelong reinforcement learning," *arXiv preprint arXiv:2001.05411*, 2020.

[7] W. C. Cheung, D. Simchi-Levi, and R. Zhu, "Reinforcement learning for non-stationary markov decision processes: The blessing of (more) optimism," in *ICML*. PMLR, 2020, pp. 1843–1854.

[8] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. Cambridge University Press, 2016.

[9] M. Eppe, P. D. H. Nguyen, and S. Wermter, "From semantics to execution: Integrating action planning with reinforcement learning for robotic causal problem-solving," *Frontiers in Robotics and AI*, vol. 6, p. 123, 2019.

[10] H. Kokel, A. Manoharan, S. Natarajan, B. Ravindran, and P. Tadepalli, "Reprel: Integrating relational planning and reinforcement learning for effective abstraction," in *ICAPS*, May 2021, pp. 533–541.

[11] M. Jin, Z. Ma, K. Jin, H. H. Zhuo, C. Chen, and C. Yu, "Creativity of ai: Automatic symbolic option discovery for facilitating deep reinforcement learning," *arXiv preprint arXiv:2112.09836*, 2021.

[12] V. Sarathy, D. Kasenberg, S. Goel, J. Sinapov, and M. Scheutz, "Spotter: Extending symbolic planning operators through targeted reinforcement learning," in *AAMAS-21*, 2021, pp. 1118–1126.

[13] F. Muhammad, V. Sarathy, G. Tatiya, S. Goel, S. Gyawali, M. Guaman, J. Sinapov, and M. Scheutz, "A novelty-centric agent architecture for changing worlds," in *AAMAS*, 2021, pp. 925–933.

[14] K. Talamadupula, G. Briggs, M. Scheutz, and S. Kambhampti, "Architectural mechanisms for handling human instructions for open-world mixed-initiative team tasks and goals," *Advances in Cognitive Systems*, vol. 5, 2017.

[15] L. Nair, J. Balloch, and S. Chernova, "Tool macgyvering: Tool construction using geometric reasoning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5837–5843.

[16] K. Takahashi, T. Ogata, S. Sugano, and G. Cheng, "Dynamic motion learning for a flexible-joint robot using active-passive motor babbling," in *The 33rd Annual Conference of the Robotics Society of Japan*, 2015, pp. 2G1–07.

[17] L. O. Ouali, C. Rich, and N. Sabouret, "Plan recovery in reactive htns using symbolic planning," in *International Conference on Artificial General Intelligence*. Springer, 2015, pp. 320–330.

[18] E. Gizzi, A. Hassan, W. W. Lin, K. Rhea, and J. Sinapov, "Toward creative problem solving agents: Action discovery through behavior babbling," in *2021 IEEE International Conference on Development and Learning (ICDL)*, 2021, pp. 1–7.

[19] T. Thai, M. Shen, N. Varshney, S. Gopalakrishnan, U. Soni, C. Baral, M. Scheutz, and J. Sinapov, "An architecture for novelty handling in a multi-agent stochastic environment: Case study in open-world monopoly," *Designing Artificial Intelligence for Open Worlds: Papers from the 2022 Spring Symposium , Virtual. AAAI Press*, 2022.

[20] K. Khetarpal, M. Riemer, I. Rish, and D. Precup, "Towards continual reinforcement learning: A review and perspectives," *arXiv preprint arXiv:2012.13490*, 2020.

[21] S. Padakandla, K. Prabuchandran, and S. Bhatnagar, "Reinforcement learning algorithm for non-stationary environments," *Applied Intelligence*, vol. 50, no. 11, pp. 3590–3606, 2020.

[22] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *CoRL*. PMLR, 2020, pp. 1094–1100.

[23] E. C. Johnson, E. Q. Nguyen, B. Schreurs, C. S. Ewulum, C. Ashcraft, N. M. Fendley, M. M. Baker, A. New, and G. K. Vallabha, "L2explorer: A lifelong reinforcement learning assessment environment," *Designing Artificial Intelligence for Open Worlds: Papers from the 2022 Spring Symposium , Virtual. AAAI Press*, 2022.

[24] Y. Yang, J. Luo, Y. Wen, O. Slumbers, D. Graves, H. B. Ammar, J. Wang, and M. E. Taylor, "Diverse Auto-Curriculum is Critical for Successful Real-World Multiagent Learning Systems," 2021. [Online]. Available: http://arxiv.org/abs/2102.07659

[25] D. Lyu, F. Yang, B. Liu, and S. Gustafson, "SDRL: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning," *AAAI 2019*, pp. 2970–2977, 2019.

[26] L. Illanes, X. Yan, R. T. Icarte, and S. A. McIlraith, "Symbolic plans as high-level instructions for reinforcement learning," *ICAPS*, vol. 30, no. Icaps, pp. 540–550, 2020.

[27] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl-the planning domain definition language," 1998.

[28] J. Hoffmann, "The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables," *Journal of artificial intelligence research*, vol. 20, pp. 291–341, 2003.

[29] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *JMLR*, vol. 21, pp. 1–50, 2020.

[30] S. Narvekar, J. Sinapov, M. Leonetti, and P. Stone, "Source task creation for curriculum learning," in *AAMAS*, 2016, pp. 566–574.

[31] P. Auer, "Using confidence bounds for exploitation exploration trade-offs," *JMLR*, vol. 3, no. Nov, pp. 397–422, 2002.

[32] S. Goel, G. Tatiya, M. Scheutz, and J. Sinapov, "Novelgridworlds: A benchmark environment for detecting and adapting to novelties in open worlds," in *AAMAS Adaptive Learning Agents (ALA) Workshop*, 2021.

[33] R. Glatt and A. H. R. Costa, "Policy reuse in deep reinforcement learning," in *AAAI*, 2017.

[34] A. Clegg, W. Yu, Z. Erickson, J. Tan, C. K. Liu, and G. Turk, "Learning to navigate cloth using haptics," in *IROS*. IEEE, 2017, pp. 2799–2805.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[36] T. K. Kim, "T test as a parametric statistic," *Korean journal of anesthesiology*, vol. 68, no. 6, p. 540, 2015.